Simd 加速 sm4 实验报告

网络空间安全创新创业实践

赵翔正 202000460090

将 simd 所用头文件包含在内

```
#include <iostream>
#include <immintrin.h>
```

初始化 ck 和 s 盒

```
const ui CK[32] = {
0x00070E15,0x1C232A31,0x383F464D,0x545B6269,
0x70777E85,0x8C939AA1,0xA8AFB6BD,0xC4CBD2D9,
0xE0E7EEF5,0xFC030A11,0x181F262D,0x343B4249,
0x50575E65,0x6C737A81,0x888F969D,0xA4ABB2B9,
0xC0C7CED5,0xDCE3EAF1,0xF8FF060D,0x141B2229,
0x30373E45,0x4C535A61,0x686F767D,0x848B9299,
0xA0A7AEB5,0xBCC3CAD1,0xD8DFE6ED,0xF4FB0209,
0x10171E25,0x2C333A41,0x484F565D,0x646B7279 };

const unsigned char SBOX[256] = {
0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C, 0x05,
0x2B, 0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99,
0x9C, 0x42, 0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC, 0x62,
0xE4, 0xB3, 0x1C, 0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F, 0xA6,
0x47, 0x07, 0xA7, 0xFC, 0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F, 0xA8,
0x68, 0x6B, 0x81, 0xB2, 0x71, 0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D, 0x35,
0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58, 0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78, 0x87,
0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27, 0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8, 0x9E,
0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5, 0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15, 0xA1,
0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD, 0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1, 0xE3,
0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29, 0x23, 0xAB, 0x0D, 0x53, 0x4E, 0x6F
```

定义加密时使用的运算

```
inline ui ROL(ui a, int n) {
    return (a << n) | (a >> (64 - n));
}

inline ui T(ui a) {
    ui t = (SBOX[a >> 24] << 24) + (SBOX[(a >> 16) & 255] << 16) + (SBOX[(a >> 8) & 255] << 8) + SBOX[a & 255];
    return  t xor ROL(t, 13) xor ROL(t, 23);
}

ui F(ui a) {
    ui t = (SBOX[a >> 24] << 24) + (SBOX[(a >> 16) & 255] << 16) + (SBOX[(a >> 8) & 255] << 8) + SBOX[a & 255];
    return t xor ROL(t, 2) xor ROL(t, 10) xor ROL(t, 18) xor ROL(t, 24);
}
```

定义生成密钥的函数，其中使用了 simd 指令进行加速

```
void genKey(ui* org) {
    __m128i K = _mm_set_epi32(org[3], org[2], org[1], org[0]);
    __m128i FK = _mm_set_epi32(0xB27022DC, 0x677D9197, 0x56AA3350, 0xA3B1BAC6);
    K = _mm_xor_si128(K, FK);
    _mm_storeu_epi32(k, K);
    for (int i = 4; i < 36; i++) k[i] = CK[i - 4];//初始化k数组
    ui xorA, xorB, t;
    for (int i = 0; i < 32; i++) {
        xorA = k[i + 1] xor k[i + 2];
        xorB = k[i + 3] xor k[i + 4];
        t = T(xorA xor xorB);
        k[i + 4] = k[i] xor t;
    }
}
```

定义加解密函数

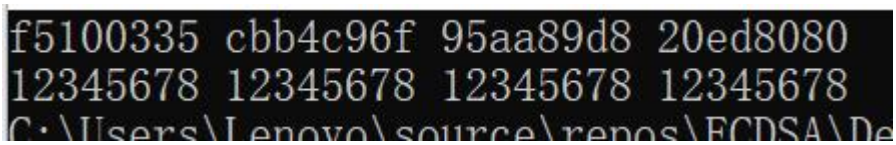
```
void encrypto(ui* msg) {
    for (int i = 0; i < 4; i++) m[i] = msg[i];
    for (int i = 4; i < 36; i++) m[i] = k[i];
    ui xorA, xorB, t;
    for (int i = 0; i < 32; i++) {
        xorA = m[i + 1] xor m[i + 2];
        xorB = m[i + 3] xor m[i + 4];
        t = F(xorA xor xorB);
        m[i + 4] = m[i] xor t;
    }
}

void decrypto(ui* msg) {
    for (int i = 0; i < 4; i++) m[i] = msg[i];
    for (int i = 4; i < 36; i++) m[39 - i] = k[i];
    ui xorA, xorB, t;
    for (int i = 0; i < 32; i++) {
        xorA = m[i + 1] xor m[i + 2];
        xorB = m[i + 3] xor m[i + 4];
        t = F(xorA xor xorB);
        m[i + 4] = m[i] xor t;
    }
}
```

运行，检验是否正确：

```
int main()
{
    ui msg[4] = { 0x12345678,0x12345678 ,0x12345678,0x12345678 };
    ui key[4] = { 0x12345678,0x12345678 ,0x12345678,0x12345678 };
    genKey(key);
    encrypto(msg);
    std::printf("%08x %08x %08x %08x", m[35], m[34], m[33], m[32]);
    for (int i = 0; i < 4; i++) {
        msg[i] = m[35 - i];
    }
    genKey(key);
    decrypto(msg);
    std::cout << std::endl;
    std::printf("%08x %08x %08x %08x", m[35], m[34], m[33], m[32]);
    return 0;
}
```

结果如下：


f5100335 cbb4c96f 95aa89d8 20ed8080
12345678 12345678 12345678 12345678

结果正确